DIGITAL SIGNAL PROCESSING MINI-PROJECT

IMAGE ENHACEMENT

JACOB J PANICKER
EE21B021

TABLE OF CONTENTS

01

INTRODUCTION

IMAGE ENHANCEMENT

Image enhancement typically refers to the process of improving the visual quality of an image. There are various techniques and tools available for image enhancement, and the choice of method depends on the specific goals and characteristics of the image.

Below are few of the techniques/methods:-

- Brightness and Contrast Adjustment
- Color Correction
- Sharpness Enhancement
- Noise Reduction
- Image Resizing
- Histogram Equalization
- Saturation Adjustment
- HDR (High Dynamic Range)
- Image Filtering
- Dehazing

In this mini-project, we'll be focusing on the noise reduction aspect, as well as grazing over the other techniques, by using filters and implementing a code in Python using Python libraries.

FACTS ABOUT THE TECHNIQUES

1. Brightness and Contrast Adjustment
   - Adjust the brightness or contrast of an image
2. Color Correction
   - Balancing the correct to correct any color cast
3. Sharpness Enhancement
   - Enhances the edges and details in the image
4. Noise Reduction
   - Removing unwanted noise signals in the image
5. Image Resizing
   - Resizing an image while maintaining the same quality
6. Histogram Equalization
   - Spreads out the pixel intensities in an image to cover the entire range
7. Saturation Adjustment
   - Adjusting the saturation of colors in the image, controlling the vividness
8. HDR (High Dynamic Range)
   - Capturing a wider range of tones/multiple exposures of the same scene
9. Image Filtering
   - Blur, vignette, sepia filters or effect addition
10. Dehazing
    - Reducing the impact of atmospheric haze in outdoor photos

02

THEORY

NOISE REDUCTION

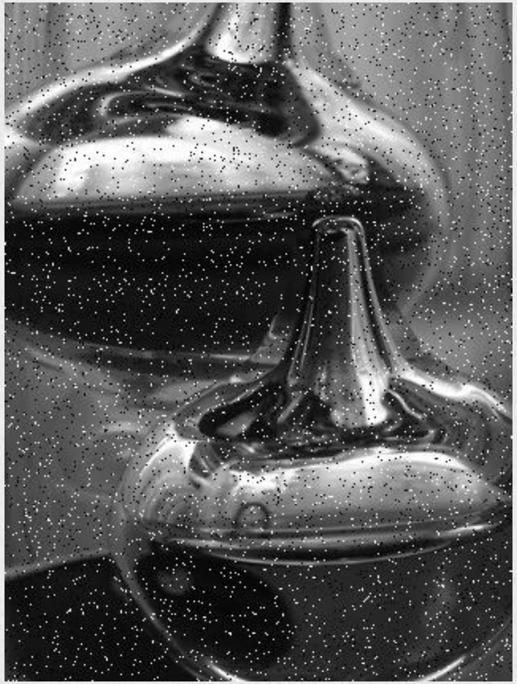
Unwanted changes in pixel values that do not belong in the scene are called noise in pictures. Sensor limitations, transmission interference, and environmental factors are among the causes of image noise. Improving an image's quality and sharpness by reducing or eliminating these undesired changes is what noise reduction is all about.

There can be different types of noises, but in terms of being more studied upon, there are broadly two:-

1. Gaussian Noise: This type of noise is characterized by random variations that follow a Gaussian (normal) distribution. It appears as random speckles and can affect both the brightness and color of individual pixels.

2. Salt and Pepper Noise: This type of noise introduces random occurrences of very bright (salt) or very dark (pepper) pixels. It simulates the appearance of random, isolated outliers in the image.

Examples for noisy images



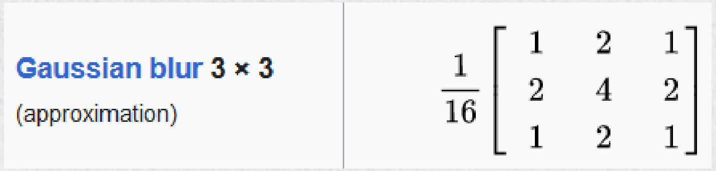
NOISE REDUCTION

The objective of noise reduction techniques is to maintain the key elements of a picture while simultaneously smoothing out or filtering out the changes that are not desired.

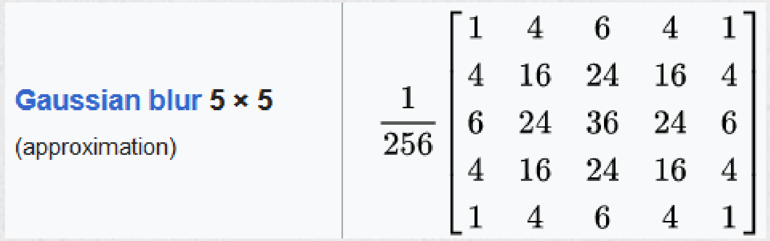
Mentioned below are two techniques:-

1. Gaussian Blur: This method involves applying a convolution operation with a Gaussian kernel to the image. The convolution blurs the image, reducing high-frequency noise while preserving the overall structure. The size of the Gaussian kernel determines the extent of smoothing.

2. Median Filtering: This method replaces each pixel value with the median value in its neighborhood. Median filtering is particularly effective at removing salt and pepper noise, as it replaces outliers with more typical pixel values.



Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$



Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$

MATHEMATICAL ANALYSIS

Generally, an image is considered as a matrix arrangement of pixels, so each pixel has co-ordinates of the form (row number, column number). Depending on the image, it can have more than one channels. A channel refers to a grayscale image component representing the intensity of light for a specific color. In a standard color image, we often have three channels corresponding to the primary colors: red, green, and blue. In this project, we'll be focusing on images with only one channel, i.e., greyscale images.

Each pixels has a value which can range from 0 to 255, where these values indicate the intensity of a pixel, so 0 indicates minimum intensity (black) and 255 indicates maximum intensity (white).

In order to make variations to the values of these pixels, we make use of a function which is called the kernel. A kernel, convolution matrix, or mask is a small matrix used for blurring, sharpening, embossing, edge detection, and more. This is accomplished by doing a convolution between the kernel and an image.

Convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other. It is the process of adding each element of the image to its local neighbors, weighted by the kernel.

MATHEMATICAL ANALYSIS

When doing the convolution, the kernel is to be placed on top of the convolution matrix such that the center element of the kernel coincides with the pixel of the image which we want to undergo the change. When doing so, a problem may arise, due to the fact that the kernel might go out of bounds of the image. In such cases, edge handling is done. Some of them are noted below:-

1. Extend: The nearest border pixels are conceptually extended as far as necessary to provide values for the convolution. Corner pixels are extended in 90° wedges. Other edge pixels are extended in lines.

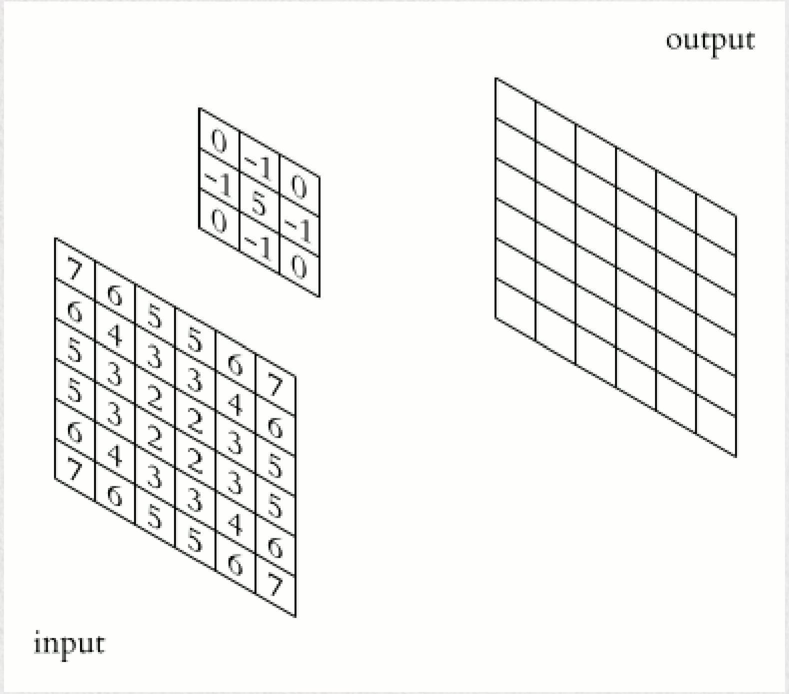
2. Wrap: The image is conceptually wrapped (or tiled) and values are taken from the opposite edge or corner.

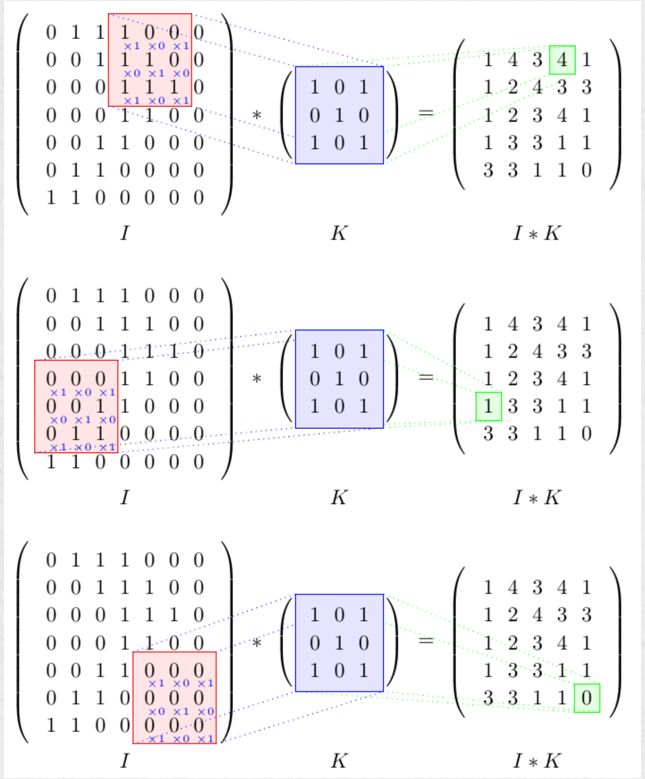
3. Mirror: The image is conceptually mirrored at the edges. For example, attempting to read a pixel 3 units outside an edge reads one 3 units inside the edge instead.

4. Constant: Use constant value for pixels outside of image. Usually black or sometimes gray is used. Generally this depends on application.

For simplicity, the technique used in this project is the constant edge handling, padding with zeroes.

MATHEMATICAL ANALYSIS





03

CODE

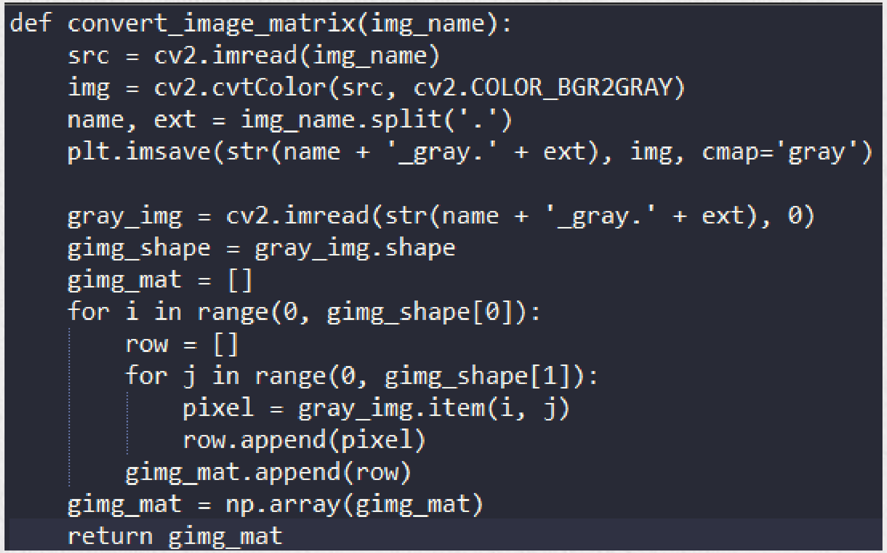
LIBRARIES USED

- Language used: Python
- numpy
- matplotlib
  - matplotlib.pyplot
- cv2
- math

1.CONVERTING TO GREYSCALE

The input picture is first gray-scaled and then transformed into a pixel matrix where each pixel's value ranges from 0 to 255 representing the intensity of that pixel at that location.

However, the conversion is only necessary for color photos and those pictures already in the grayscale mode can be directly converted into the image matrix.



```
def convert_image_matrix(img_name):
    src = cv2.imread(img_name)
    img = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    name, ext = img_name.split('.')
    plt.imsave(str(name + '_gray.' + ext), img, cmap='gray')

    gray_img = cv2.imread(str(name + '_gray.' + ext), 0)
    gimg_shape = gray_img.shape
    gimg_mat = []
    for i in range(0, gimg_shape[0]):
        row = []
        for j in range(0, gimg_shape[1]):
            pixel = gray_img.item(i, j)
            row.append(pixel)
        gimg_mat.append(row)
    gimg_mat = np.array(gimg_mat)
    return gimg_mat
```

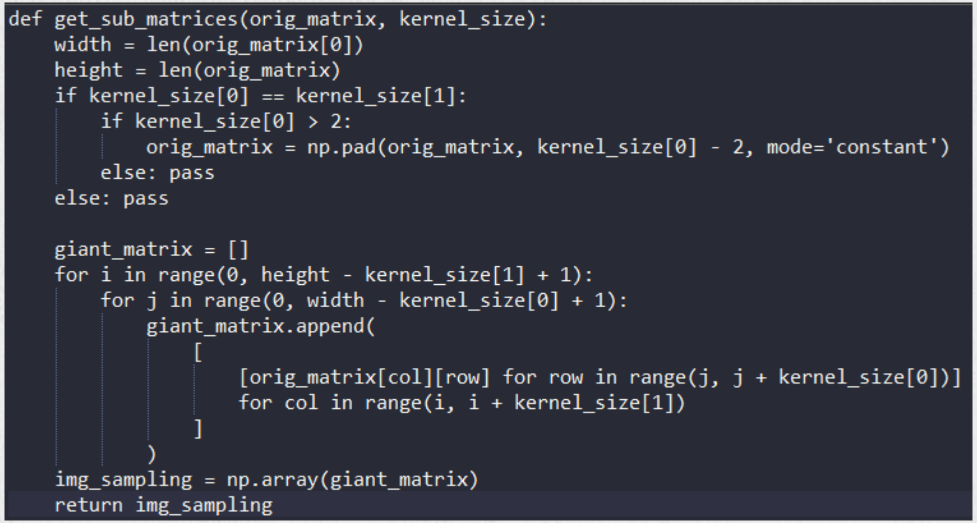
2.EDGE HANDLING

Just as discussed before, the problem of edge handling arises in this situation.

The amount of edge handling required also depends on the kernel size, so care must be taken there too. Most of the kernels are of the size 3, but size 5 also exists.

(Size N means NxN matrix)

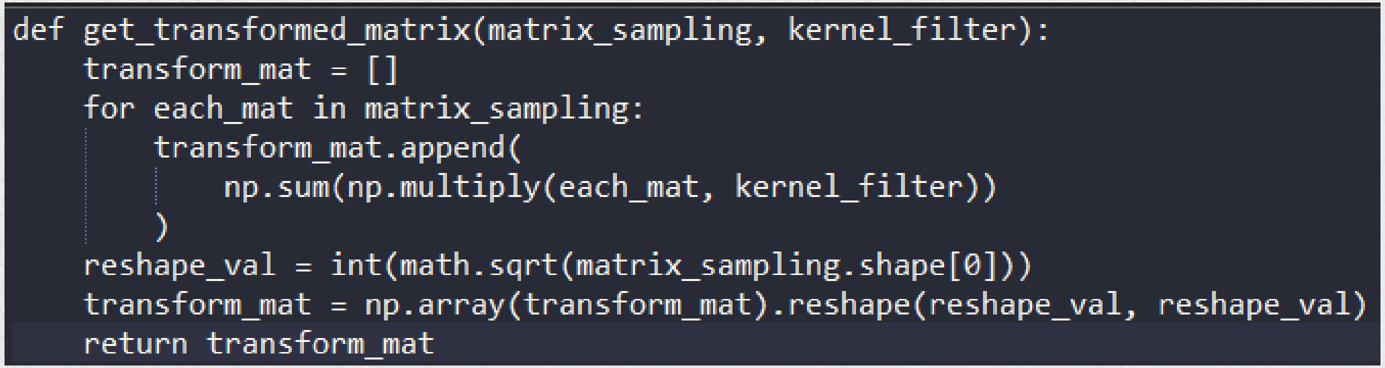
At the end of this stage, we'll get an output matrix which is then further used for convolution with the kernel function.



```python
def get_sub_matrices(orig_matrix, kernel_size):
    width = len(orig_matrix[0])
    height = len(orig_matrix)
    if kernel_size[0] == kernel_size[1]:
        if kernel_size[0] > 2:
            orig_matrix = np.pad(orig_matrix, kernel_size[0] - 2, mode='constant')
        else: pass
    else: pass

    giant_matrix = []
    for i in range(0, height - kernel_size[1] + 1):
        for j in range(0, width - kernel_size[0] + 1):
            giant_matrix.append(
                [
                    [orig_matrix[col][row] for row in range(j, j + kernel_size[0])]
                    for col in range(i, i + kernel_size[1])
                ]
            )
    img_sampling = np.array(giant_matrix)
    return img_sampling
```

3.CONVOLUTION

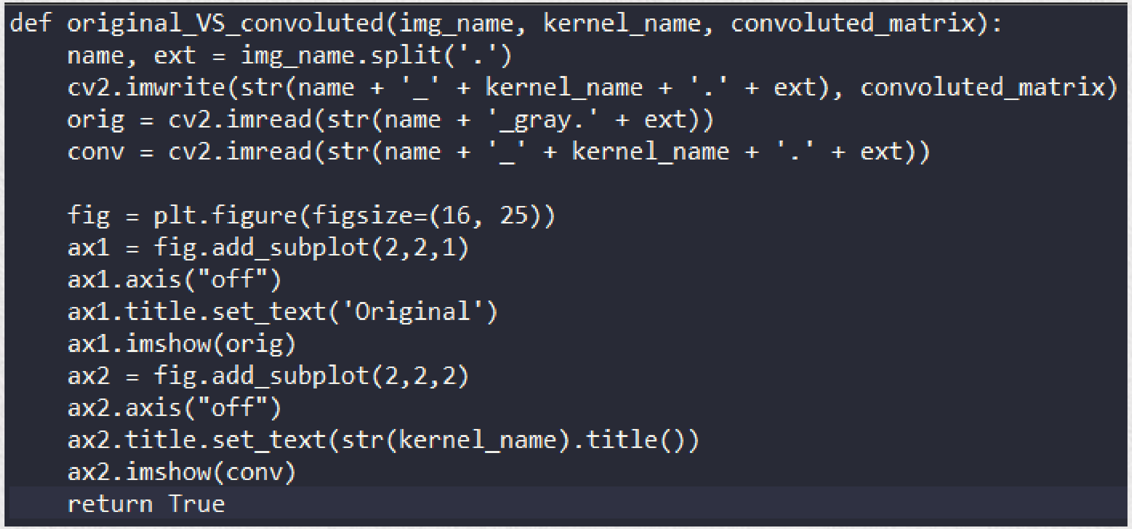
Convolution of the image matrix with the kernel matrix. Note that the orientation of the kernel with respect to the transformed image matrix should be like what was discussed earlier. (Slide 12)



```
def get_transformed_matrix(matrix_sampling, kernel_filter):
    transform_mat = []
    for each_mat in matrix_sampling:
        transform_mat.append(
            np.sum(np.multiply(each_mat, kernel_filter))
        )
    reshape_val = int(math.sqrt(matrix_sampling.shape[0]))
    transform_mat = np.array(transform_mat).reshape(reshape_val, reshape_val)
    return transform_mat
```

4. FINAL OUTPUT

The final segment deals with the comparison of the original or grayscale version of the input picture with its final filtered output. The final and the grayscale image will be saved in the directory the program was saved in.



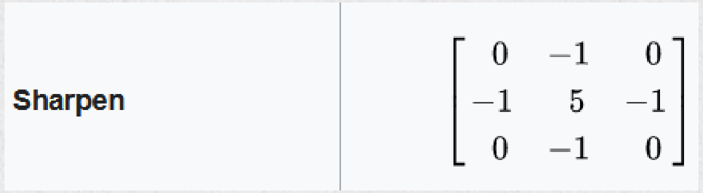
```python
def original_VS_convoluted(img_name, kernel_name, convoluted_matrix):
    name, ext = img_name.split('.')
    cv2.imwrite(str(name + '_' + kernel_name + '.' + ext), convoluted_matrix)
    orig = cv2.imread(str(name + '_gray.' + ext))
    conv = cv2.imread(str(name + '_' + kernel_name + '.' + ext))

    fig = plt.figure(figsize=(16, 25))
    ax1 = fig.add_subplot(2,2,1)
    ax1.axis("off")
    ax1.title.set_text('Original')
    ax1.imshow(orig)
    ax2 = fig.add_subplot(2,2,2)
    ax2.axis("off")
    ax2.title.set_text(str(kernel_name).title())
    ax2.imshow(conv)
    return True
```

04

EXTRAS

SHARPENING

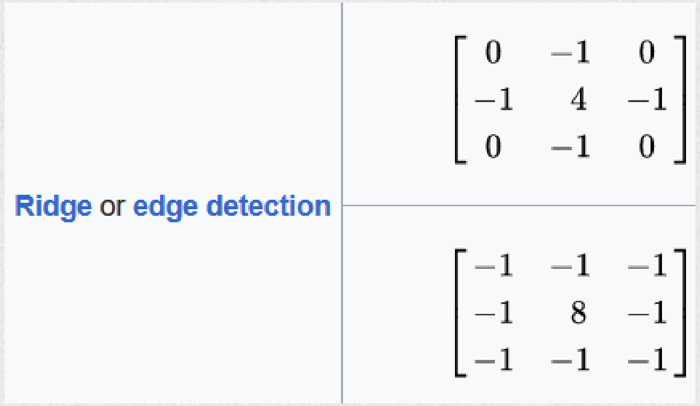
Digital photographs may be enhanced using a technique called image sharpening to make them seem more detailed. The process of sharpening makes an image’s edges stand out more. Images with low quality at the margins are drab. Background and edges are almost indistinguishable. In contrast, a sharpened picture is one where the viewer can make out the edges.



Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

IMAGE EDGE DETECTION

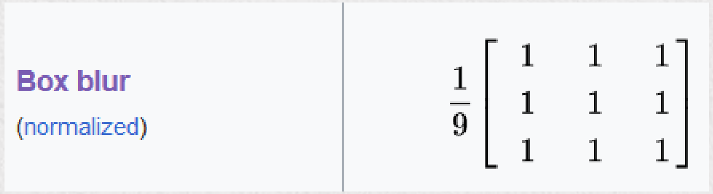
By using edge detection, viewers are able to scrutinize an image's characteristics for a notable shift in grayscale. With this texture, we can see where one area of the picture ends and another begins. It maintains a picture's structural integrity while decreasing the quantity of data in the image. This is a popular technique for deep learning feature extraction from images.



Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

BOX BLUR

Computer graphics and image processing often make use of box blur, a straightforward method for blurring images. In it, inside a predetermined rectangular region called the ”box,” the average color value of each pixel is calculated by adding up the values of all the pixels immediately around it. The method softens and reduces the appearance of distinct characteristics by reducing high-frequency noise and smoothing out small details. The blur effect is proportional to the box’s size; bigger boxes result in more noticeable blurring. Since it doesn’t need a lot of processing power, box blur is a go-to for real-time uses like video games and editing multimedia, where simple and fast blurring is needed.

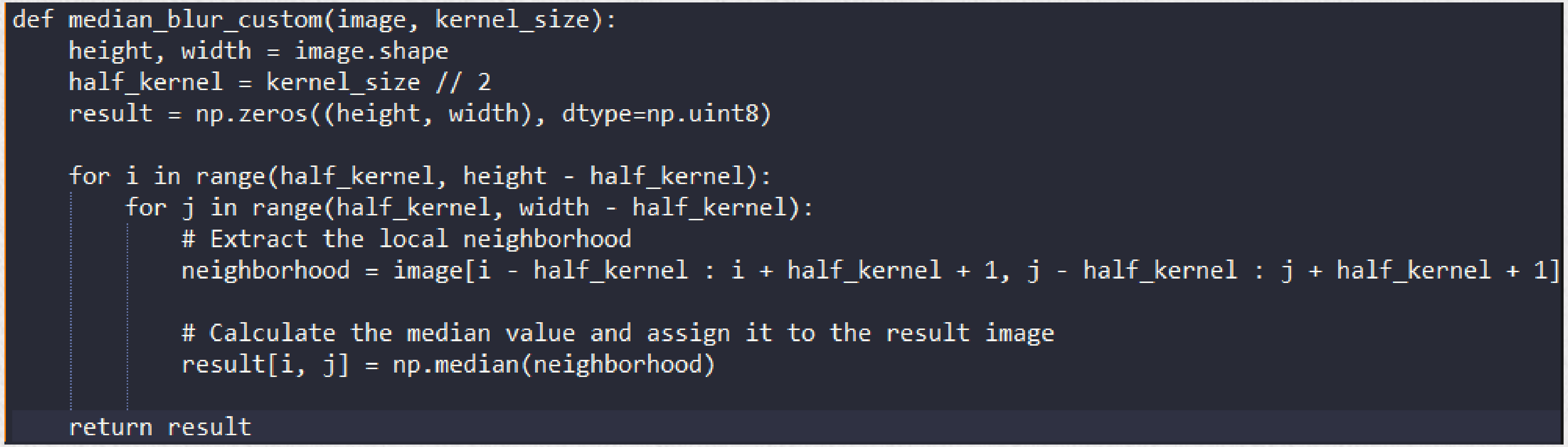


Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

MEDIAN BLUR CODE

As discussed before, there are kernels for each purpose such as sharpening, blurring and edge detection, but median blur doesn't. Instead, it follows a mathematical procedure.

Median blur includes taking the average value of the neighboring pixel intensities and replaces the pixel which is getting interfered by the noise. This is very much effective against salt and pepper noise.

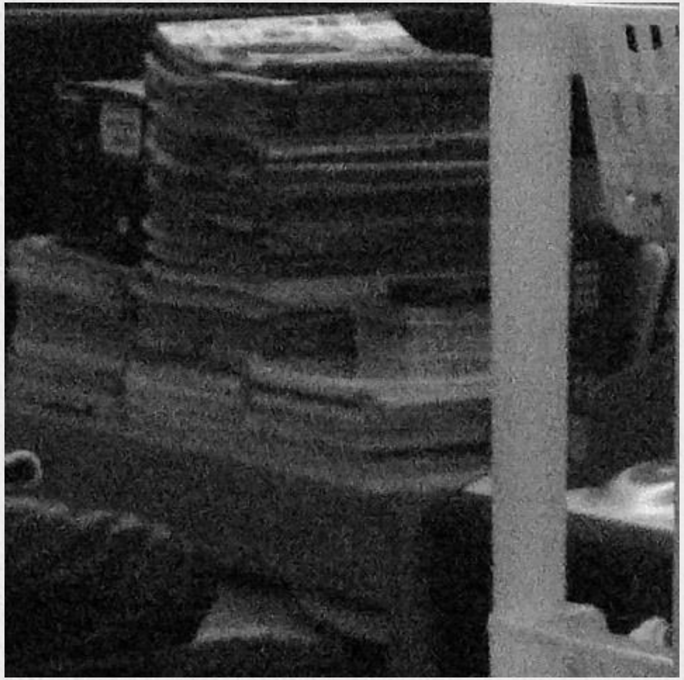


```python
def median_blur_custom(image, kernel_size):
    height, width = image.shape
    half_kernel = kernel_size // 2
    result = np.zeros((height, width), dtype=np.uint8)

    for i in range(half_kernel, height - half_kernel):
        for j in range(half_kernel, width - half_kernel):
            # Extract the local neighborhood
            neighborhood = image[i - half_kernel : i + half_kernel + 1, j - half_kernel : j + half_kernel + 1]

            # Calculate the median value and assign it to the result image
            result[i, j] = np.median(neighborhood)

    return result
```

05

DEMOS

GAUSSIAN BLUR



Original Image

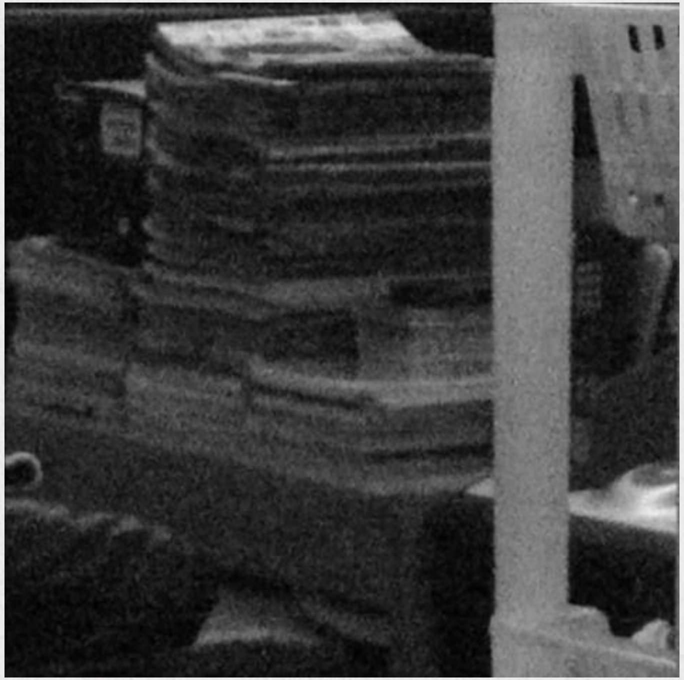


Image after Gaussian Blur

BOX BLUR

Original Image

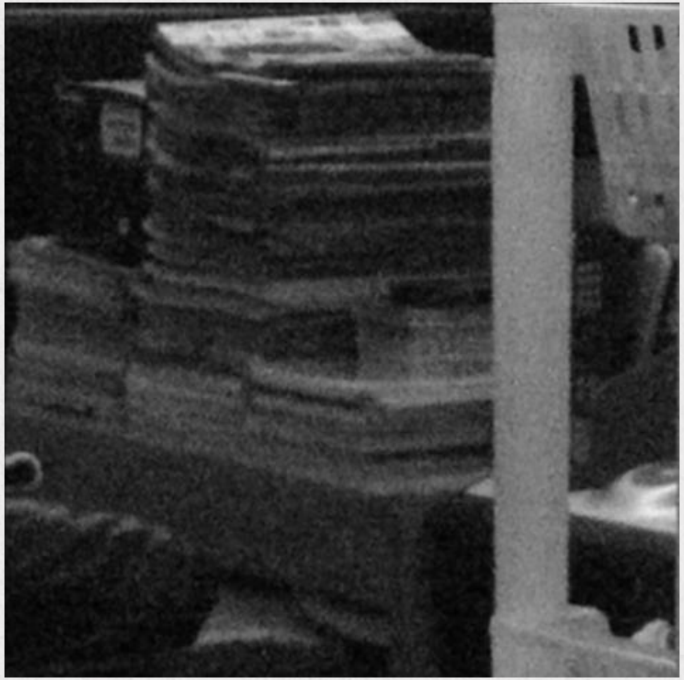
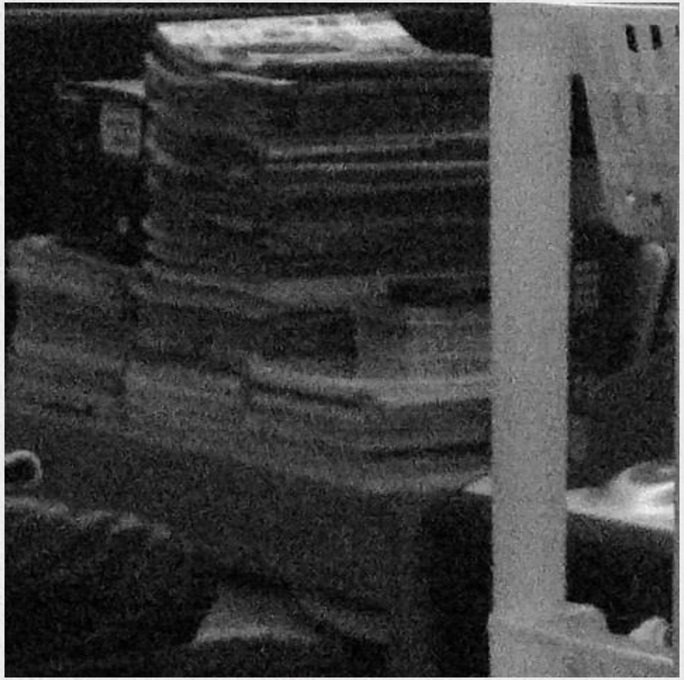


Image after Box Blur

MEDIAN BLUR



Original Image

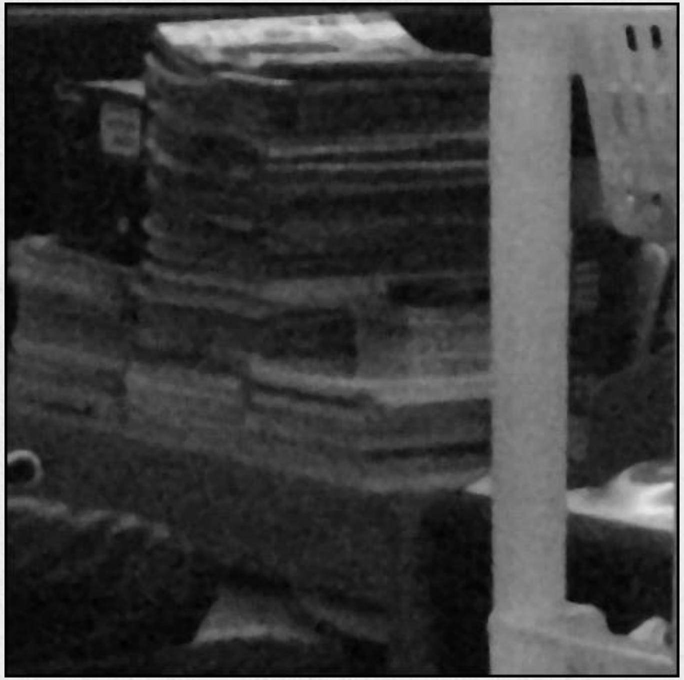
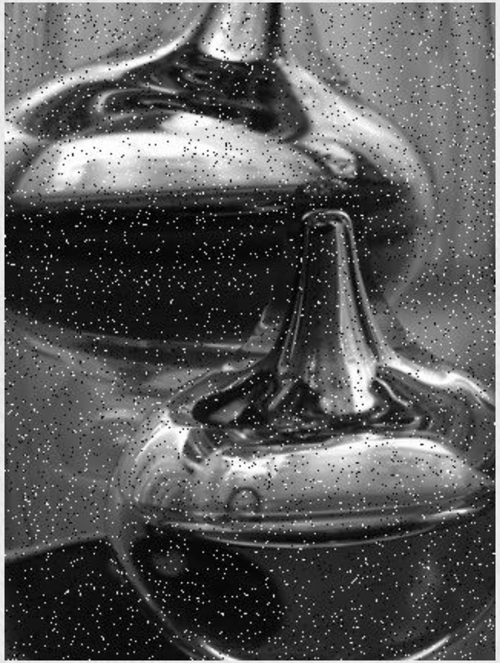


Image after Median Blur

MEDIAN BLUR


Original Image

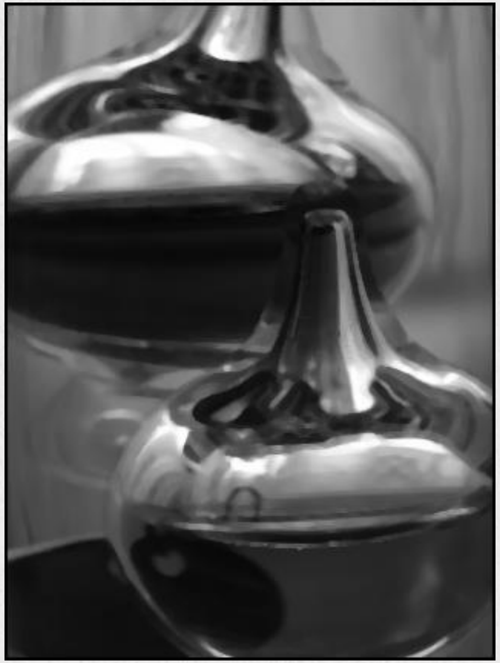


Image after Median Blur

SHARPENING

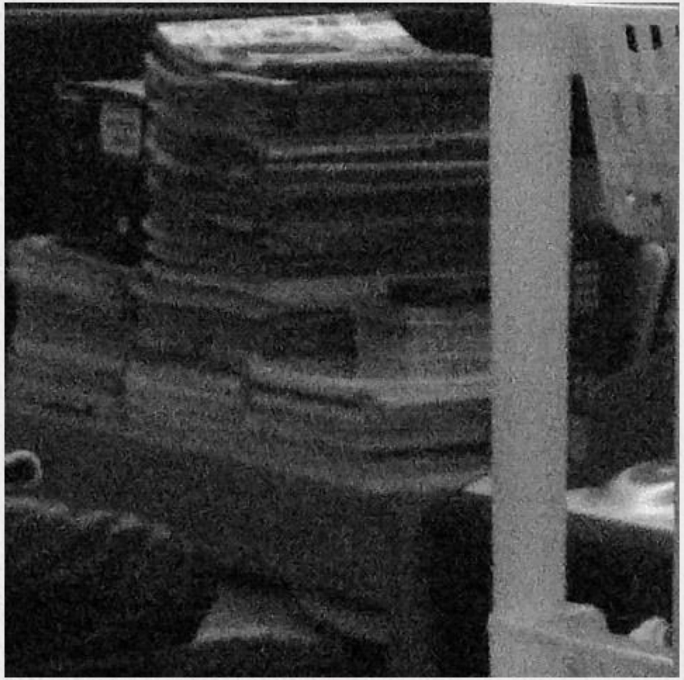
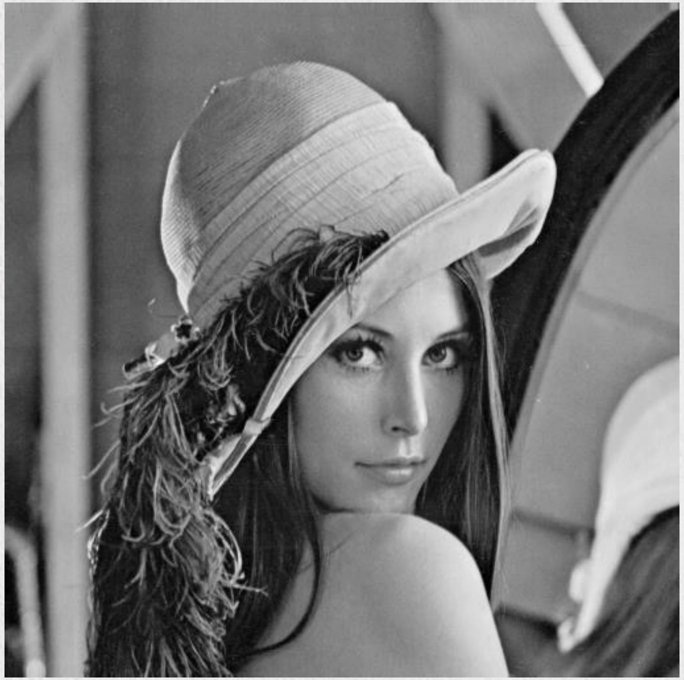

Original Image

Image after Sharpening

SHARPENING


Original Image

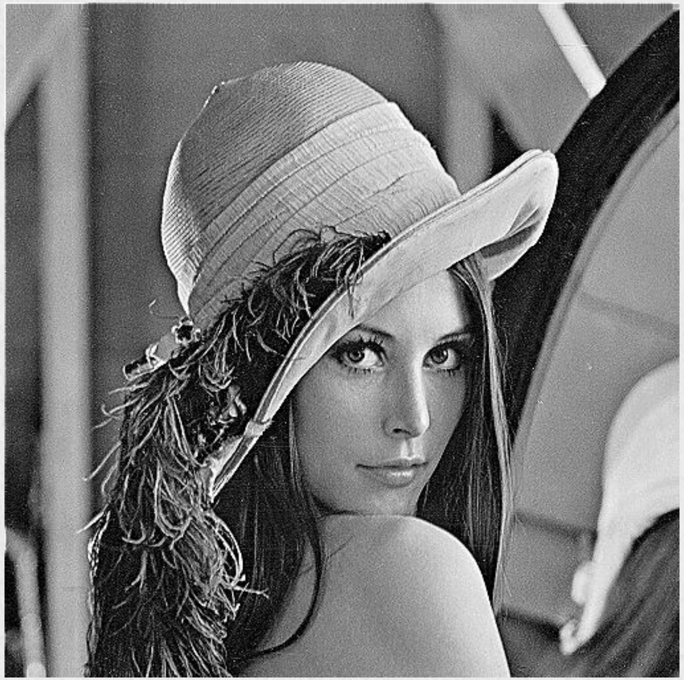


Image after Sharpening

EDGE DETECTION

Original Image

Image after Edge Detection

EDGE DETECTION

Original Image

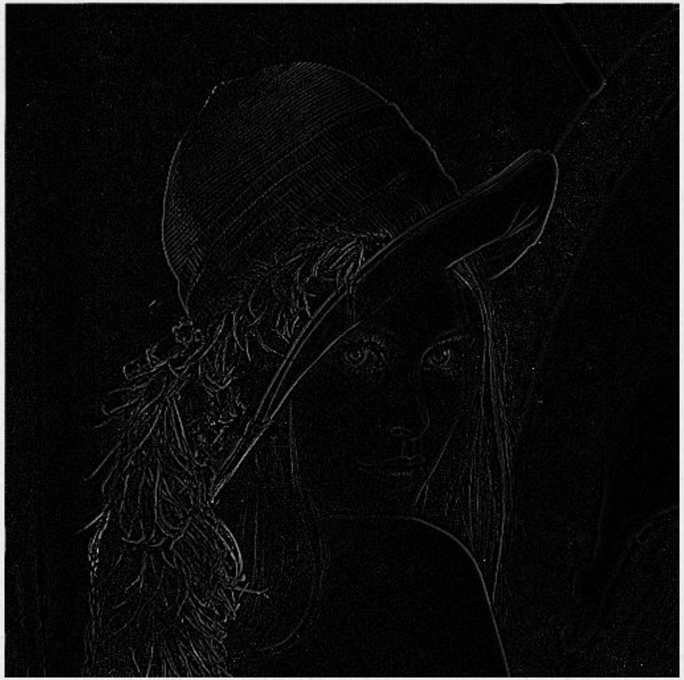


Image after Edge Detection

06

REFERENCES

REFERENCES

- MATLAB Documentation
- Library file documentations
    - Numpy
    - OpenCV
    - Matplotlib
- GeeksforGeeks
- The Medium
- Wikipedia articles
- ScienceDirect

THANKS!

Fonts & colors used

This presentation has been made using the following fonts:

Lexend
(https://fonts.google.com/specimen/Lexend)

Space Grotesk
(https://fonts.google.com/specimen/Space+Grotesk)

#191919 #f0efef #ffffff #4619bb #20124d

slidesgo